AMENDMENTS TO THE SPECIFICATION:

Please delete page 28, line 5 – page 29, line 13.

Please replace the paragraph beginning on page 1, line 5, with the following:

The present invention relates to a data processing apparatus, system and method for generating program code for translating high level code into instructions for one or more target processors and, separately, to a data processing apparatus system and method for running such program code. In particular, but not exclusively, the program code forms a virtual machine, for example a JAVA software virtual machine, for the one or more target processors.

Please replace the paragraph beginning on page 2, line 17, with the following:

In order to aid application development, and to re-use applications to run on different host processors, it is desirable that the application code is transportable between different host processors. This provides for re-use of whole applications, or parts thereof, thereby increasing the speed of development of applications for new processors and indeed increasing the speed of development of new applications themselves. This may be achieved by means of program code which runs on a host processor and is capable of translating high level program code into operation code or instructions for the host processor. The program code provides a virtual machine for a host processor, enabling it to implement application software written in an appropriate high level language. An example of such translating program code is the JAVA software programming language developed by Sun Microsystems, Inc. (JAVA is a trademark of Sun Microsystems, Inc).

Such program code, when running on an appropriate host processor is known as a JAVA software Virtual Machine.

Please replace the paragraph beginning on page 2, line 30, with the following:

Although examples of embodiments of the present invention will be described with reference to JAVA software and JAVA software Virtual Machines, embodiments in accordance with the invention are not limited to the JAVA programming language but may be implemented using other suitable programming languages for forming virtual machines.

Please replace the paragraph beginning on page 3, line 1, with the following:

A feature of a virtual machine is that it provides for the dynamic loading of applications onto embedded processing systems. This is an extremely useful feature. Typically, applications are already embedded within a processing system. It is difficult to dynamically download an application or to patch an existing application onto an embedded processing device. However, virtual machines, such as JAVA software enabled, provide the possibility of enabling dynamic loading of a complete application that could be written by a third party and available on a remote server, for example. Moreover, distribution and maintenance costs are reduced since it is possible to dynamically interact with the embedded system via the virtual machine. Due to JAVA software application program interfaces APIs standardisation, the configuration and the profiles, reference [1], for compatibility of applications can be ensured if the JAVA software platform on the embedded system is compliant with the standardisation.

Please replace the paragraph beginning on page 3, line 17, with the following:

Security features are also available within JAVA software to identify a trusted code which is dynamically downloaded through a network and to preserve the availability of the embedded system.

Please replace the paragraph beginning on page 3, line 21, with the following:

Another feature of JAVA software is that the hardware architecture heterogeneity management may be masked. A major advantage of such a feature is that it reduces the software development costs of an application. Embedded processors typically are highly diverse and have specific capabilities and capacities directed to the needs of the system or appliance in which they are embedded. This would generally give rise to a high cost of application development. However, because of the portable nature of JAVA software code between JAVA software Virtual Machines, the cost of integrating a new hardware architecture, for example, merely relies on developing a new JAVA software Virtual Machine. Another important feature is that the transparent exploitation of a multi-processor architecture can be achieved by a JAVA software Virtual Machine, without any change of the application code.

Please replace the paragraph beginning on page 3, line 32, with the following:

Two known JAVA software virtual machines are the JWORKS [6] and KVM, the JAVA software virtual machine of J2MF [7]. JWORKS is part of personal JAVA software virtual machine distribution on VXWORKS real-time operating systems. As VXWORKS is designed to be integrated on a large range of hardware platforms and because JWORKS

is, at least from the operating system point of view, an application respecting VXWORKS APIs, JWORKS could be executed on a large range of hardware platforms. Nevertheless, the integration of JWORKS on a new embedded system is limited to the VXWORKS porting ability, without any consideration of the JAVA software virtual machine. However, as will be described in detail later, a JAVA software virtual machine must take care of many different aspects. In light of this, JWORKS is unable to provide the best trade-off for a dedicated embedded system, since it does not take into account the different requirements of target host processors in the embedded system.

Please replace the paragraph beginning on page 4, line 16, with the following:

J2ME is a Sun Java platform for small embedded devices. KVM is the JAVA software virtual machine of J2ME. It supports 16 and 32 bits CISC and RISC processors, and generates a small memory footprint and can keep the code in a memory area of about 128 KB. It is written for a ANSI C compiler with the size of basic types well defined (e.g. character on 8 bits, long on 32 bits). This is why it is difficult to port the KVM onto another compiler (for example the TI DSP C55x C family compilers support characters on 16 bits), without re-writing all the JAVA software virtual machines. Additionally, an optional data alignment can only be obtained for 64 bit data. Other alignments are handled by the C compiler. Moreover, there is no possibility to manage a heterogeneous multiprocessor without re-writing C structures (due to data representation conversion). It is also not possible to tune a trade-off between memory and access costs without re-writing substantially all the parts of the JAVA software virtual machine.

Please replace the paragraph beginning on page 5, line 14, with the following:

Viewed from one aspect, the present invention teaches how to build a virtual machine (JAVA software Virtual Machine) which may be compatible for several embedded systems. In this regard, the Virtual Machine comprises program code modules, each module providing a particular function and optimised for a particular host processor.

Please replace the paragraph beginning on page 9, line 15, with the following:

Embodiments of the present invention provide development of virtual machines, for example JAVA software virtual machines using modular adaptability such that it is possible to support in a straight forward manner any kind of data representation on a target processor or processors. Additionally, language mapping can be added to the development environment for supporting different compilers. Furthermore, each module can define its own data alignment as well as also introducing transparent data representation conversion to support heterogeneous architecture. The user memory may also be optimised.

Please replace the paragraph beginning on page 10, line 31, with the following:

The following illustrative examples of embodiments of the invention will be described with reference to JAVA software and JAVA software Virtual Machines. However, embodiments of the invention are not limited to JAVA software programming languages or virtual machines.

Please replace the paragraph beginning on page 11, line 1, with the following:

A JAVA software Virtual Machine allows an embedded processing system to be abstracted out of the embedded system as far as an application programmer is concerned. However, this means that a JAVA software Virtual Machine has to take into account different aspects of an embedded processor system, such as the hardware architecture, the tool chain available for the hardware, the hardware operating system as well as the application requirements.

Please replace the paragraph beginning on page 11, line 12, with the following:

The integration of these four different aspects represents a significant challenge in design and implementation of a JAVA software Virtual Machine since there are many options and choices which may be taken for abstracting the different aspects of the embedded processing system, to arrive at a solution to a particular embedded processing system.

Please replace the paragraph beginning on page 12, line 16, with the following:

Each processor may define its own data representation capabilities, for example from 8 bits to 128 bits and possibly more in future processing devices. For efficient operation, a JAVA software Virtual Machine must be capable of manipulating byte code which is adapted for the particular data representation of the target processor. The availability of a floating point support such as 32- or 64-bit floating point in a processor may also be utilised by JAVA software Virtual Machine to implement the **float** or **double** JAVA software data types. Additionally, the registers available in a target processor may

be exploited, or at least a sub-set of them, to optimise JAVA software stack performance. For example, one register can be used for the representation of the JAVA software stack pointer. Another consideration of the hardware architecture is the memory alignment issues, for example whether it is constant, proportionate to size, proportioned with threshold or some other such criterion. Additionally, the memory access cost has to be taken into account in order to efficiently arrange object fields within the JAVA software interface. Furthermore, if the embedded system comprises a multi-processor system, then the existence of homogenous or heterogeneous data representation (/ENDIAN) data size alignment has to be correctly managed in order to share JAVA software objects or internal JAVA software Virtual Machine data structures between the different processors in the multi-processor system.

Please replace the paragraph beginning on page 13, line 5, with the following:

An embedded system may well have many different types of memories, for example RAM, DRAM, FLASH, local RAM, which have to be taken into account as regards their particular performance characteristics. Additionally, the size of each memory has to be considered. For example, a local variable pool can be stored in local RAM whilst class files may be arranged in FLASH memory. Another consideration is that with a shared memory multi-processors, the JAVA software Virtual Machine must manage homogenous or heterogeneous address space to correctly share JAVA software objects or JAVA software Virtual Machine internal structure. The cache architecture, such as whether it has one or two levels or a particular type of flash, must also be taken into account in order to properly implement JAVA software synchronisation and the volatile attributes of an object field.

Please replace the paragraph beginning on page 13, line 16, with the following:

For mobile or portable applications, an important aspect of the processor system is the use by the JAVA software Virtual Machine of energy aware instruction sets such that the byte code generated for the JAVA software Virtual Machine minimise the system energy consumption.

Please replace the paragraph beginning on page 13, line 20, with the following:

Another aspect for consideration is that processors, or families of processors, are typically associated with a tool chain. A tool chain provides a series of functions and processes implemented in the target processor/s instruction code which may be called via a JAVA software Virtual Machine. Typically, each hardware platform makes various languages available for implementation upon it, for example C++, optimised C or assembler, which can realise optimisations about memory consumption, use of an optimised instruction set, pre-processor code optimisation and the use of register, inlining calls, 64 bit support amongst other things. Whilst it is evident that the use of a ANSI C compiler would increase the portability of an implementation, it should be borne in mind that other languages are available for hardware platforms. The particular capabilities of a compiler, as provided by their tool chain, may be implemented within a JAVA software Virtual Machine for that processor.

Please replace the paragraph beginning on page 13, line 32, with the following:

Another aspect which needs to be considered when developing a JAVA software Virtual Machine for a target processor/s, is the operating system that the target system runs under. Generally, an operating system has some good and some bad properties and, consequently, may be better for certain types of embedded system and not others. Particularly, operating systems tend to be designed to address a particular type of application and use. For example, POSIX operating systems such as LINUX, WIN-CE, SYMBIAN, are designed for general applications, a real-time operating system such as VX WORKS, NUCLEUS, and RTEMS are designed for real-time applications and dedicated processor kernel operating systems such as DDP BIOS SPOX, PALM OS for specific embedded applications or digital signal processing. Each of the foregoing operating systems and applications have differences which impact upon the implementation of a JAVA software application program interface. For example, to be compliant to RTSJ reference [2] a real-time operating system should be used.

Please replace the paragraph beginning on page 14, line 17, with the following:

A further aspect to be considered is the application requirements for processing systems. This is particularly important for an embedded system which would typically be directed to a particular type of application. As is well-known, a JAVA software application requires the use of JAVA software application program interfaces. Depending upon application needs, several application program interfaces have been defined, for example J2MF for embedded devices and J2SE for desk-top computers. A JAVA software Virtual Machine has to provide, or not, the relevant application program interfaces to support, or not, a compliant application. Additionally, new processor devices can be masked through application program interfaces and, therefore, a JAVA software Virtual Machine needs to be able to deal with low-level implementations of the devices. An example would be a JAVA software Virtual Machine supporting a blue tooth

communications network protocol for which an appropriate application program interface would have to be defined. Application program interfaces are not necessarily the sole preserve of the application programmer, but may well be the result of standard specifications such as may be derived by a JAVA software community process group.

Please replace the paragraph beginning on page 14, line 31, with the following:

Figure 2 illustrates the process flow for implementing an application using a JAVA software Virtual Machine. The process starts at step 120 where an application in JAVA software source code is developed and written. That application source code is compiled in a JAVA compiler at step 122 which converts the application source code into an architecture neutral object file format thereby encoding a compiled instruction sequence at step 124, in accordance with the JAVA software Virtual Machine specification. The compiled instruction sequence at step 124 consists of a plurality of byte codes. The byte codes are executed by the JAVA software Virtual Machine at step 125, which translates the byte codes into processor instructions for implementation on processor 100 at step 128.

Please replace the paragraph beginning on page 15, line 13, with the following:

As discussed above, a large number of criteria have to be taken into account when designing a JAVA software Virtual Machine for embedded processing systems. The design of a JAVA software Virtual Machine is complex, particularly if certain goals are to be achieved. Namely, those goals are to minimise the importing cost of the JAVA software Virtual Machine onto a new embedded processing system; to obtain the best trade off between application needs, embedded system processing constraints and embedded

system features; and to adapt the JAVA software Virtual Machine to features of new hardware and new applications.

Please replace the paragraph beginning on page 15, line 21, with the following:

The present applicant has addressed the problems and difficulties encountered in developing JAVA software Virtual Machines to meet the foregoing design criteria by developing a modular JAVA software Virtual Machine architecture. The term "modular" means at least two associated things. Firstly, it refers to a specification of all individual different software parts for a JAVA software Virtual Machine and, secondly, a way to agglomerate these many parts, written in separate languages, with maximum transparency in order to provide a modularised environment for generating a modular JAVA software virtual machine. Additionally, the applicant has invented an architecture in which it is possible to transparently access a particular module from other modules. Consequently, the architecture provides for the investigation of several implementation choices for a module for one specific embedded system by testing the implementation system with regard to other modules and best design choice. The choice of implementation may be done in accordance with different features of the embedded system such as its hardware architecture, its tool chain, operating system and application requirements.

Please replace the paragraph beginning on page 16, line 8, with the following:

By designing and implementing a modular JAVA software Virtual Machine, the JAVA software Virtual Machine may be implemented for a particular embedded system. And the trade-off between the various attributes of the system can be optimised for the particular target processors and application.

Please replace the paragraph beginning on page 16, line 13, with the following:

Referring now to Figure 3, there is illustrated an example of a development environment such as may be provided by a software tool, for developing a modular JAVA software Virtual Machine (MDE). The MDE 200 provides a series of tools to support the modular design of a JAVA software Virtual Machine. Three inputs are sent to the MDE in order to generate the sources and glues. First, an interface definition language (IDL) specification 204 which describes declarations of JAVA software Virtual Machine services and types utilising a processor independent language is input. Additionally, various program code modules containing implementations of JAVA software Virtual Machine services and types for one of a number of languages which can be run on target processors can be input. Finally, alignment definitions 208 describing the alignment constraints for different target processors, together with their respective access costs are also input to MDE 200.

Please replace the paragraph beginning on page 16, line 24, with the following:

A tool chain 210 compiles sources and glues 212 to generate program code to form a JAVA software Virtual Machine suitable for the processing device or specific embedded system for which the JAVA software Virtual Machine is targeted. The tool chain 210 generates a JAVA software virtual machine for each processor of a multi-processor system. Each JAVA software virtual machine comprises its own modules, and preferably does not share modules with other JAVA software virtual machines. In this regard, each JAVA software virtual machine is independent of the other. The choice of modules for each JAVA software virtual machine depends on the design criteria input 220.

Please replace the paragraph beginning on page 16, line 32, with the following:

Describing the specification for the modular JAVA software Virtual Machine interface definition language 204 in more detail, the IDL describes services and data types independently of the language implementation of the JAVA software Virtual Machine. The service comprises a function name with a return type and all data types used for that service and the direction of its parameters. The type itself is composed of other types or structured types such as structures, unions and arrays, with services to create and free instances of the type and parameters to initialise such instances.

Please replace the paragraph beginning on page 17, line 11, with the following:

The implementation modules, 206, each comprise service or type implementations having common or shared implementation characteristics or knowledge. When a particular module M1, M2, M3 or M4 is selected to be part of the JAVA software Virtual Machine, all services and type implementations inside the selected module are also selected. A scheduler module is also included as part of the implementation modules 206. The scheduler module undertakes load balancing to determine whether a task is mapped to a particular one or other of a plurality of processing devices in an embedding system, e.g. DSP or MPU.

Please replace the paragraph beginning on page 18, line 10, with the following:

The modular development environment 200 comprises three main components: the module chooser 214, the type manager 216 and several language mappings 218. In order to be able to generate different aspects or versions of a JAVA software Virtual Machine for

different target host processors within a multi-processor system, several different modules are configured to implement the same services and types, yet for different processor languages. In this case, each module is described by a set of keywords. The module chooser 214 selects from the plurality of modules M1, M2, M3 and M4 the most accurate one or ones for a specific embedded processing system in accordance with design criteria 220 input thereto. The design criteria comprises a list of weighted keywords.

Please replace the paragraph beginning on page 19, line 14, with the following:

Each module of the JAVA software virtual machine may be of a so-called open design. Examples of module design addressing the four different points described above in relation to hardware through to application requirements will now be described. Extensions to module designs are described for each point.

Please replace the paragraph beginning on page 19, line 30, with the following:

A software tool in accordance with an embodiment of the invention can be utilised by an implementer to write special modules to efficiently use a hardware's potential for a target processor. For example, a dedicated stack module could use a local double access memory or chip to optimise the performance of the energy consumption of the JAVA software virtual machine.

Please replace the paragraph beginning on page 20, line 8, with the following:

Adding a new tool chain may be achieved by adding a new language mapping in the MDE. Thus, new language support may be added, for example an assembler for a specific target processor, or the JAVA software language itself in connection with a native interface support. Similarly, it can be possible to exploit or compensate for a tool chain or compiler features, e.g. lining support.

Please replace the paragraph beginning on page 20, line 22, with the following:

With regard to software requirements, the modular approach permits a module to add JAVA software virtual machine functionality or native methods to respective application requirements by implementing, completing, tracking or intercepting all services or types. For example, a module could implement services and types to provide a class loader. Optionally, a module could complete a type. For example, a thread module could add a private mutax on object type and class type without modifying other modules which use these types. Hooks can be added on any service, so that for example a benchmark/profile/debug module could trace every service in the JAVA software virtual machine.

Please replace the paragraph beginning on page 20, line 31, with the following:

Any service may be intercepted, for example a module could intercept the class access service to add multiprocessing by cloning static fields and monitors of classes as described in reference [4]. Since the specification of the JAVA software platform is subject to evolution, for example a JAVA software virtual machine compliant with CLDC

specification has to be CDC compliant (class load support, floating support, byte code verifier support, etc.), new or extensions of a JAVA software virtual machine specification can significantly change the design of several elements of the JAVA software virtual machines. The initial effort for obtaining a JAVA software virtual machine for a first target is significant since the first decomposition of the modules is non trivial and the initial design can take a long time. However, once this effort has been expended, then further advantage may be taken of the modularity decomposition in order to generate JAVA software virtual machines for further target processors.

Please replace the paragraph beginning on page 21, line 14, with the following:

Embodiments in accordance with the present invention for developing and providing a JAVA software virtual machine require the use of several tools, and are limited in respect of the coding restrictions in the implementation of module services. Advantageously, new tools may be developed in order to improve the implementation of modules.

Please replace the paragraph beginning on page 21, line 19, with the following:

The integration of hardware and software for the design of respective modules requires strong skills both in hardware design, especially on an analysis of the side effects introduced by a treatment on architecture, in tool chains (it is necessary to support a new language, or to adapt MDE tool according to a language facility), in the operating system (in order to manage the JAVA software virtual machine efficiently through the operating system by choosing the best facilities available) and algorithmically for op code realisation. The foregoing challenges exist for other non-modular JAVA virtual machines.

Embodiments in accordance with the present invention, differ from conventional JAVA software module machines and their design in that it provides the possibility for the designer to exploit their skills in order to determine the best trade-off for the JAVA software virtual machine, to run on a multi-processor system environment or optionally on a single processor.

Please replace the paragraph beginning on page 21, line 31, with the following:

An embodiment in accordance with the present invention comprises a JAVA software virtual machine designed using a modular approach.

Please replace the paragraph beginning on page 22, line 1, with the following:

In a preferred embodiment, the JAVA software virtual machine is decomposed in several modules classified in six categories as illustrated in Table 1 of Figure 4. The six categories are arithmetic, JAVA software frame, control flow, object, class and miscellaneous. Each module is given a name indicative of the function that it is performed. For example, the module which handles the representation, operations, and conversion of JAVA software integers, is termed "integer". In a particular embodiment, the hardware supported is a PENTIUM 32 bit core processor family. Additionally, the Texas Instruments DSPc55x family, which is a 16 bit word addressable processor, is also supported. In the preferred embodiment, most modules are written in ANSI C compiler, a few are written with the GNU C features and TI DSP: C55x. Modules may also be implemented in PENTIUM and DSP assembly language, and also in DSP optimised C compiler for increasing the efficiency of bit code execution.

Please replace the paragraph beginning on page 22, line 32, with the following:

A DSP is a dedicated processor for a digital signal processing. In spite of its limitation when used as a general microprocessor, its large usage in embedded systems and also some interesting features means that the running of a partial or a full JAVA software virtual machine on such a hardware platform is not a meaningless proposal. In the following section the hardware features of the TI C55x family, and its tool chain characteristics in operating systems are briefly described. Further details may be found from reference [3].

Please replace the paragraph beginning on page 23, line 3, with the following:

In the preferred embodiment, the interpretation engine of the illustrative JAVA software virtual machine is referred to as a motor module. The motor module is in charge of decoding the byte code in order to call the appropriate function to perform the op code operation associated with the byte code. In a first implementation, a classical loop is used to fetch one op code, to decode it with a C switch statement (or similar statement for another implementation language) and to branch to the right piece of code. In a second implementation, so-called threaded code, reference [5], is used to translate, prior to execution, the original byte code by sequence of addresses which point directly to op code implementation. At the end of each code implementation, a branch to the next address is performed and so on. Such an implementation avoids the decoding phase of the switch solution described above, but causes an increase or expansion in the amount of codes.

Please replace the paragraph beginning on page 24, line 23, with the following:

Table 2 illustrated in Figure 5 summarises the number of cycles used to execute one loop of a JAVA class-file to compute fibonacci numbers and the number of memory bits to store the byte code (translated or not).

Please replace the paragraph beginning on page 25, line 9, with the following:

As can be clearly illustrated by the foregoing, the gain obtained by an optimisation for one particular processor is not necessary the same as the other. Therefore, it is optimistic to believe that optimisation can be achieved in a completely portable code. Thus, compiling a portable source of a JAVA software virtual machine without fully understanding the target processor or processors could result in poor performance or unwitting and undesired memory expansion.

Please replace the paragraph beginning on page 25, line 27, with the following:

Embodiments of the invention provide an easy way to add modules between two other ones transparently. Moreover, modules may be easily adapted to JAVA software virtual machines for embedded systems, such that the JAVA software virtual machine can be adapted to new features depending upon the application requirement. This permits the taking into account of hardware and software evolution. As an example, the management of multimedia applications taking into account energy consumption may be included as a module or modules within embodiments of the invention, reference [8]. Moreover, the reusability of existing module implementation limits the adaptation of few modules only

on a new embedded system and that reduces development cost. Finally, the independence between modules increases its ability of overall JAVA software virtual machine, and also the maintenance cost of the JAVA software virtual machine software.